The Complexity of $3SAT_N$ and the $\mathcal{P}$ versus $\mathcal{NP}$ Problem

Ruijia Liao

Abstract

We introduce the $\mathcal{NP}$-complete problem $3SAT_N$ and extend Tovey's results to a classification theorem for this problem. This classification theorem leads us to generalize the concept of truth assignments for SAT to aggressive truth assignments for $3SAT_N$. All these aggressive truth assignments are pseudo-algorithms. We combine algorithm, pseudo-algorithm and diagonalization method to study the complexity of $3SAT_N$ and the $\mathcal{P}$ versus $\mathcal{NP}$ problem. The main result is $\mathcal{P} \neq \mathcal{NP}$.

1. Introduction

In computational complexity theory, the Boolean satisfiability problem (SAT) is a decision problem, in which the instance is a Boolean expression written using only AND, OR, NOT, variables, and parentheses. The question is: given the expression, is there some assignment of $true$ and $false$ values to the variables that make the entire expression true? SAT is the first known $\mathcal{NP}$-complete problem, proven by Stephen Cook in 1971 [6]. Independently in 1973, Leonid Levin showed that a variant of the tiling problem is $\mathcal{NP}$-complete [9]. In 1972, Richard Karp proved that several other problems were also $\mathcal{NP}$-complete [10]. In particular, the Boolean satisfiability problem remains $\mathcal{NP}$-complete even if all expressions are written in conjunction normal form with 3 variables per clause (3-CNF), yielding the $3SAT$ problem. We can define the $kSAT$ problem in a similar way. Let (r, s)-SAT be the class of instances with exactly r distinct variables per clause and at most s occurrences per variable. In 1984 [13], Craig Tovey proved that every instance of (r, r)-SAT is satisfiable and $(3, 4)$-SAT is $\mathcal{NP}$-complete. Indeed, using these $(3, s)$-SAT for $s = 1, \cdots, 4$ and the results in [13], we can easily classify all instances in $3SAT$ by polynomial time in each instance's length.

In this paper, we use Tovey's idea in [13] to classify the $\mathcal{NP}$-complete problem $3SAT_N$. We prove that this classification takes polynomial time. With this classification theorem, we introduce the concept of aggressive truth assignment. An aggressive truth assignment is a polynomial time pseudo-algorithm. Using these pseudo-algorithms, we endow some set of algorithms with a metric and then introduce Cauchy sequences among them. The

Cauchy sequences of algorithms in this paper are essentially the Cauchy sequences of pseudo-algorithms. Like the role of Cauchy sequences of rational numbers in real number theory, the Cauchy sequences of algorithms allow us to use pseudo-algorithms to approximate some algorithms. In 1874, Cantor [4] established that real numbers are uncountable. Sixteen years later, he proved his theorem again using diagonal argument [5]. Surprisingly, by analyzing computations of some pseudo-algorithms, we can use Cantor's diagonalization method to prove that there are uncountably many algorithms under some assumption. It contradicts the fact that there are only countably many algorithms (see e.g. [8]). Therefore, the assumption must be false. The role of the classification theorem is crucial, which prevents applying the new argument to $2SAT$ and other $\mathcal{P}$ problems in SAT to generate any contradiction.

In 1975, T.Baker, J.Gill, and R.Solovary [3] introduced the following relativized worlds: there exist oracles A and B, such that $\mathcal{P}^A \neq \mathcal{NP}^A$ and $\mathcal{P}^B = \mathcal{NP}^B$. They also pointed out that the relativizing method could not solve the $\mathcal{P}$ versus $\mathcal{NP}$ problem. In the early 1990's, A.Razborov and S.Rudich [11] defined a general class of proof techniques for circuit complexity lower bounds, called natural proofs. At the time all previously known circuit lower bounds were natural, and circuit complexity was considered a very promising approach for resolving $\mathcal{P} = \mathcal{NP}$. However, Razborov and Rudich showed that, if certain kinds of one-way functions exist, then no natural proof method can distinguish between $\mathcal{P}$ and $\mathcal{NP}$. Although one-way functions have never been formally proven to exist, most mathematicians believe that they do. Thus it is unlikely that natural proofs alone can resolve $\mathcal{P} = \mathcal{NP}$. In 1992, A.Shamir [12] used a new non-relativizing technique to prove $IP = PSPACE$. However, in 2008, S.Aaronson and A.Wigderson [1] showed that the main technical tool used in the $IP = PSPACE$ proof, known as arithmetization, was also insufficient to resolve $\mathcal{P} = \mathcal{NP}$. In this paper, each pseudo-algorithm can be explicitly expressed by Algorithm 1 and Algorithm 2, however, it is not an algorithm. It takes finite steps to partially evaluate any $\eta \in 3SAT$ and, most importantly, it is different from any oracle and arithmetization. Since the new argument combines algorithm, pseudo-algorithm and diagonalization method, it circumvents relativization, natural proofs and algebrization.

In Section 2, we give some definitions and notations, and describe an algorithm. We prove that $3SAT_N$ is an $\mathcal{NP}$-complete problem in Section 3. We extend Tovey's results in [13] to a classification theorem in $3SAT_N$ and define another algorithm in Section 4. We generalize the concept of truth assignment to aggressive truth assignment in Section 5 based on the Classification Theorem. We define composition of two or more aggressive truth assignments and investigate, $\mathcal{TA}^\infty$, a set of all aggressive truth assignments under this operation. We introduce the concept of distance between any two elements in $\mathcal{TA}^\infty$ and endow it with a metric. In Section 6, we extend this metric concept to $< f >$, a set of algorithms generated by algorithm f and aggressive truth assignments, in which we can define Cauchy sequences. We also introduce the definition of pseudo-algorithm. We discuss equivalence of algorithms and pseudo-algorithms in Section 7. Any two elements in $\mathcal{TA}^1$ are equivalent. However, $\mathcal{TA}^2$ has infinitely many equivalence classes. Combining algorithm, pseudo-algorithm and diagonalization method, we prove $\mathcal{P} \neq \mathcal{NP}$ in section 8.

2. Preliminary

Let $SAT(n)(x_1, \cdots, x_n)$ be the set of all expressions in SAT in which each element uses exactly n variables and their negations $\{x_1, \cdots, x_n, \neg x_1, \cdots, \neg x_n\}$. From the definition, $SAT(n) \cap SAT(n+1) = \emptyset$ and $\cup_{n=3}^{\infty} SAT(n) \subseteq SAT$. For $r \leq n$ and $1 \leq s$, let (r,s)-$SAT(n) = (r,s)$-$SAT \cap SAT(n)$. We can show that, for any $\eta \in SAT$, there exists a polynomial time algorithm in the length of η to find the integer n such that η is generated by variables and their negations $\{x_{i_1}, \cdots, x_{i_n}, \neg x_{i_1}, \cdots, \neg x_{i_n}\}$ where $i_1 < \cdots < i_n$. Furthermore, there exists a linear time map ϕ_{map}, such that $\phi_{map}(x_{i_k}) = x_k, \phi_{map}(\neg x_{i_k}) = \neg x_k$, for $k = 1, \cdots, n$ and $\tilde{\eta} = \phi_{map}(\eta) \in SAT(n)$. Thus, if $\eta \in (r,s)$-SAT, then $\tilde{\eta} = \phi_{map}(\eta) \in (r,s)$-$SAT(n)$ for some n. Obviously, $\tilde{\eta}$ is satisfiable if and only if η is satisfiable, and the numbers of clauses of $\tilde{\eta}$ and η are the same. For any $\eta \in SAT(n)$, the above algorithm is trivial and the map ϕ_{map} can be viewed as the identical map. In the following discussion, for convenience, we use ϕ_{map} to represent the above algorithm and the map.

The map x_i^* from $\{x_j, \neg x_j\}$ to $\{true, false, undef\}$ is defined as $x_i^*(x_i) = true, x_i^*(\neg x_i) = false$, $x_i^*(x_j) = undef$ and $x_i^*(\neg x_j) = undef$ if $i \neq j$. The map $\neg x_i^*$ is defined in a similar way. An atomic truth assignment e_i is defined as $e_i = x_i^*$ or $e_i = \neg x_i^*$. The negation operator can be applied to e_i too, i.e., if $e_i = x_i^*$, $\neg e_i = \neg x_i^*$, and if $e_i = \neg x_i^*$, $\neg e_i = x_i^*$.

Let

$$\begin{aligned} \eta &= (y_{1,1} \vee \cdots \vee y_{1,j_1}) \wedge (y_{2,1} \vee \cdots \vee y_{2,j_2}) \wedge \cdots \wedge \\ &\quad (y_{m,1} \vee \cdots \vee y_{m,j_m}) \in SAT(n), \end{aligned} \tag{1}$$

where $y_{i,j} = x_k$ or $\neg x_k$ for some k, the subscript $\{i,j,k\} \in \{1, 2, \cdots, n\}$. A truth assignment $e_1 e_2 \ldots e_n$ is defined as

$$\begin{aligned} e_1 e_2 \ldots e_n(\eta) &= (e_{1,1}(y_{1,1}) \vee \cdots \vee e_{1,j_1}(y_{1,j_1})) \wedge \\ &\quad (e_{2,1}(y_{2,1}) \vee \cdots \vee e_{2,j_2}(y_{2,j_2})) \wedge \cdots \wedge \\ &\quad (e_{m,1}(y_{m,1}) \vee \cdots \vee e_{m,j_m}(y_{m,j_m})) \in \{true, false\}. \end{aligned} \tag{2}$$

Note that this definition can be used to describe an algorithm to evaluate any $\eta \in SAT(n)$.

Algorithm 1.

```
for $k = 1, 2, \cdots, m$
    evaluate clause $(y_{k,1} \vee \cdots \vee y_{k,j_k})$ as follows:
    for $l = 1, 2, \cdots, j_k$
        for $p = 1, 2, \cdots, n$
            if $e_p(y_{k,l}) = undef$, continue to the inner loop, i.e., try next $e_{p+1}$;
            if $e_p(y_{k,l}) = false$ and $p < n$ and $l < j_k$, continue to the middle loop, i.e., try
                next $y_{k,l+1}$;
            if $e_p(y_{k,l}) = false$ and ($p = n$ or $l = j_k$), return $false$;
            if $e_p(y_{k,l}) = true$
                if $k < m$, continue to the outer most loop, i.e., try next clause;
                if $k = m$, return $true$.
```

The elementary steps of Algorithm 1 are the evaluations $e_i(y_j)$, for $i, j = 1, \cdots, n$, and returning $true$ or $false$. It is a polynomial time algorithm.

From the definition, a truth assignment $e_1e_2\cdots e_n$ is determined by each atomic truth assignment e_i, it is independent of their order. However, for convenience, it always takes ascending order. For any integers $m \geq n \geq 3$, we can apply the truth assignment $e_1e_2\cdots e_m$ to any instance of $SAT(n)$. If $m > n$, we use $e_1, e_2, \cdots, e_n$ and ignore $e_{n+1}, \cdots, e_m$. Similarly, we can define a generalized truth assignment $e_1e_2\cdots e_n\cdots$ and apply it to any instance of $SAT(n)$ for any integer $n \geq 3$. However, the generalized expression is defined by finite information. For example, for any given integer $k > 0$,

$$\begin{aligned} e_1e_2\cdots e_k\cdots &= e_1e_2\cdots e_k x^*_{k+1}x^*_{k+2}x^*_{k+3}x^*_{k+4}\cdots, \text{ or} \\ & \quad e_1e_2\cdots e_k x^*_{k+1}\neg x^*_{k+2}x^*_{k+3}\neg x^*_{k+4}\cdots, \text{ or} \\ & \quad e_1e_2\cdots e_k \neg x^*_{k+1}\neg x^*_{k+2}\neg x^*_{k+3}\neg x^*_{k+4}\cdots. \end{aligned}$$

The first expression is called the positive extension of $e_1e_2\cdots e_k$ and the last one is called the negative extension of $e_1e_2\cdots e_k$. A generalized truth assignment $e_1e_2\cdots e_n\cdots$ is called negative if $e_n = \neg x^*_n$ for any integer $n \geq 1$. Note that for any truth assignment $e_1e_2\cdots e_n$, there are countably many generalized truth assignments associated with it.

Example 1. Let $e_1 = x^*_1, e_2 = \neg x^*_2, e_3 = \neg x^*_3$, $e_4 = x^*_4$, $\eta_1 = (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$, and $\eta_2 = (x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$, then

$$\begin{aligned} e_1e_2e_3e_4(\eta_1) &= (e_1(\neg x_1) \vee e_2(\neg x_2) \vee e_3(\neg x_3)) \wedge (e_1(\neg x_1) \vee e_2(x_2) \vee e_3(x_3)) \\ &= (false \vee true \vee e_3(\neg x_3)) \wedge (e_1(\neg x_1) \vee e_2(x_2) \vee e_3(x_3)) \\ &= (e_1(\neg x_1) \vee e_2(x_2) \vee e_3(x_3)) \\ &= (false \vee false \vee false) \\ &= false, \end{aligned}$$

and

$$\begin{aligned} e_1e_2e_3e_4(\eta_2) &= (e_1(x_1) \vee e_3(x_3)) \wedge (e_1(\neg x_1) \vee e_3(\neg x_3)) \\ &= (true \vee e_3(x_3)) \wedge (e_1(\neg x_1) \vee e_3(\neg x_3)) \\ &= (e_1(\neg x_1) \vee e_3(\neg x_3)) \\ &= (false \vee true) \\ &= true. \end{aligned}$$

3. An $\mathcal{NP}$-complete Problem

Let η be an instance of $kSAT$. A clause $y_{i_1} \vee y_{i_2} \vee \cdots \vee y_{i_k}$ of η is called a tautological clause if there are i_α, i_β and variable x such that $y_{i_\alpha} = x$ and $y_{i_\beta} = \neg x$. A clause $y_{i_1} \vee y_{i_2} \vee \cdots \vee y_{i_k}$

of η is called a full clause if it has k distinct variables or their negations. In other words, a full clause has no repeated variable. An expression is called a normal expression if it satisfies the following conditions: (1) it has no tautological clause, (2) it has no repeated clause and (3) each clause is full. Let

$$kSAT_N = \{\eta \mid \eta \in kSAT \text{ and } \eta \text{ is a normal expression}\}. \tag{3}$$

We have the following:

Theorem 1. $3SAT_N$ is $\mathcal{NP}$-complete.

Proof. First, we show that $3SAT_N$ is in $\mathcal{NP}$. A nondeterministic polynomial time Turing machine can guess a truth assignment to a given expression $\eta \in 3SAT_N$ and accept if the assignment satisfies η. Next, we prove that any given $\eta \in 3SAT$ can be polynomial time in the length of η reduced to $\tilde{\eta} \in 3SAT_N$. For any $\eta \in 3SAT$,

(1) If $\eta = \tilde{\eta} \wedge \theta$, where θ is a tautological clause, then $\tilde{\eta}$ is satisfiable if and only if η is satisfiable. Remove all tautological clauses. Let η_1 be the result expression.

(2) If $\eta_1 = \tilde{\eta}_1 \wedge \theta \wedge \theta$, where θ is a repeated clause. Let $\overline{\eta}_1 = \tilde{\eta}_1 \wedge \theta$, then $\overline{\eta}_1$ is satisfiable if and only if η_1 is satisfiable. If $\overline{\eta}_1$ has any repeated clause, repeat this subprocess for $\eta_1 = \overline{\eta}_1$. Let η_2 be the result expression.

(3) If η_2 has a clause with repeated variable, remove one or two these repeated variables to get a new clause without any repeated variable, repeat this subprocess for all clauses with repeated variable. Let η_3 be the result expression.

(4) If $\eta_3 = \tilde{\eta}_3 \wedge \theta$, where clause $\theta = x$ or $\theta = x \vee y$. If $\theta = x$, we can force x to be true by means of the clauses below:

$$\begin{aligned}
&(x \vee a_{i,j}^{(x)} \vee b_{i,j}^{(x)}),\ j = 1, 2,\\
&(d_{i,j}^{(x)} \vee \neg a_{i,j}^{(x)} \vee \neg b_{i,j}^{(x)}), (d_{i,j}^{(x)} \vee \neg a_{i,j}^{(x)} \vee b_{i,j}^{(x)}), (d_{i,j}^{(x)} \vee a_{i,j}^{(x)} \vee \neg b_{i,j}^{(x)}),\ j = 1, 2,\\
&(\neg d_{i,1}^{(x)} \vee \neg d_{i,2}^{(x)} \vee \neg x).
\end{aligned}$$

In order to make the above nine clauses satisfiable, x must be true. Suppose that $x = false$, then $\neg d_{i,1}^{(x)} = true$ or $\neg d_{i,2}^{(x)} = true$ by the last clause above. If $\neg d_{i,j}^{(x)} = true$, then the trueness of all three clauses $(d_{i,j}^{(x)} \vee \neg a_{i,j}^{(x)} \vee \neg b_{i,j}^{(x)})$, $(d_{i,j}^{(x)} \vee \neg a_{i,j}^{(x)} \vee b_{i,j}^{(x)})$ and $(d_{i,j}^{(x)} \vee a_{i,j}^{(x)} \vee \neg b_{i,j}^{(x)})$ implies that $a_{i,j}^{(x)} \vee b_{i,j}^{(x)} = false$, and if $(x \vee a_{i,j}^{(x)} \vee b_{i,j}^{(x)}) = true$, we must have $x = true$. This contradicts our assumption. Let κ be the wedge of the above nine clauses. Let $\overline{\eta}_3 = \tilde{\eta}_3 \wedge \kappa$, i.e., replace clause θ with the wedge of the above nine clauses. Then $\overline{\eta}_3$ is satisfiable if and only if η_3 is satisfiable.

If $\theta = x \vee y$, we can force $x \vee y$ to be true by means of the clauses below:

$$\begin{aligned}
&(x \vee y \vee a^{(x,y)}),\\
&(d^{(x,y)} \vee \neg a^{(x,y)} \vee \neg b^{(x,y)}), (d^{(x,y)} \vee \neg a^{(x,y)} \vee b^{(x,y)}), (d^{(x,y)} \vee a^{(x,y)} \vee \neg b^{(x,y)}),\\
&(\neg d^{(x,y)} \vee x \vee y).
\end{aligned}$$

In order to make the above five clauses satisfiable, $x \vee y$ must be true. Suppose that $x \vee y = false$, then $\neg d^{(x,y)} = true$ by the last clause above. If $\neg d^{(x,y)} = true$, then the trueness of all three clauses $(d^{(x,y)} \vee \neg a^{(x,y)} \vee \neg b^{(x,y)})$, $(d^{(x,y)} \vee \neg a^{(x,y)} \vee b^{(x,y)})$ and $(d^{(x,y)} \vee a^{(x,y)} \vee \neg b^{(x,y)})$ implies that $a^{(x,y)} \vee b^{(x,y)} = false$, and if $(x \vee y \vee a^{(x,y)}) = true$, we must have $x \vee y = true$. This contradicts our assumption. Let κ be the wedge of the above five clauses. Let $\overline{\eta}_3 = \tilde{\eta}_3 \wedge \kappa$, i.e., replace clause θ with the wedge of the above five clauses. Then $\overline{\eta}_3$ is satisfiable if and only if η_3 is satisfiable. If $\overline{\eta}_3$ has any non-full clause, repeat this subprocess for $\eta_3 = \overline{\eta}_3$. Let η_4 be the result expression.

Now expression η_4 is a normal expression and η_4 is satisfiable if and only if η is satisfiable. Clearly, all subprocesses (1), (2), (3) and (4) together take polynomial time in the length of η. This completes the proof of Theorem 1.

For $n \geq 3, n \geq k, n \geq r$ and $s \geq 1$, define

$$\begin{aligned} &kSAT_N(n) = kSAT(n) \cap kSAT_N, \\ &SAT_N(n) = SAT(n) \cap SAT_N, \\ &(r,s)\text{-}SAT_N(n) = (r,s)\text{-}SAT(n) \cap SAT_N(n). \end{aligned} \tag{4}$$

We can define (r,s)-SAT_N in a similar way. We prove that $(3,4)$-SAT_N is $\mathcal{NP}$-complete in next section.

4. A Classification Theorem

In this section, we prove the following classification theorem:

Theorem 2. For every instance η of $3SAT_N$, one of the following statements is true:
(1) $\eta \in (3,1)$-SAT_N and η is satisfiable,
(2) $\eta \in (3,2)$-SAT_N and η is satisfiable,
(3) $\eta \in (3,3)$-SAT_N and η is satisfiable,
(4) $\eta \in (3,4)$-SAT_N or
(5) η can be reduced to $\tilde{\eta} \in (3,4)$-SAT_N in polynomial time in the length of η.
Moreover, checking if $\eta \in \cup_{s=1}^{3}(3,s)$-$SAT_N$ takes polynomial time in the length of η.

Proof. By the definition, $\cup_{s=1}^{\infty}(3,s)$-$SAT \subset 3SAT$. Since $(3,4)$-SAT is $\mathcal{NP}$-complete [13], for $s > 4$, any $\eta \in (3,s)$-SAT can be reduced to $\tilde{\eta} \in (3,4)$-SAT in polynomial time. We show that this reduction also transforms $\eta \in (3,s)$-SAT_N to $\tilde{\eta} \in (3,4)$-SAT_N in polynomial time. Thus, for every instance η of $3SAT_N$, $\eta \in \cup_{s=1}^{4}(3,s)$-$SAT_N$ or (5) is true. The second part of (3) is the special case of Theorem 2.4 [13]. We just need to prove the second parts of (1) and (2). By the definition of (r,s)-SAT_N, the second part of (1) is trivial. For the second part of (2), from the assumption, instance η has at least three variables, and at least one of them with occurrence number 2, say x_1, x_2 or x_3. Let $\tilde{\eta} = \eta \wedge (x_1 \vee x_2 \vee x_3)$, then $\tilde{\eta} \in (3,3)$-SAT_N and $\tilde{\eta}$ is satisfiable. Thus, η is satisfiable as well.

For the last statement of Theorem 2, suppose that η is generated by variables and their negations $\{x_{i_1}, \cdots, x_{i_n}, \neg x_{i_1}, \cdots, \neg x_{i_n}\}$ where $i_1 < \cdots < i_n$. First inspect a variable and its negation $\{x_{i_1}, \neg x_{i_1}\}$, let c be the number of their occurrences. If $c > 3$, $\eta \notin \cup_{s=1}^{3}(3, s)\text{-}SAT_N$, this process completes. Otherwise, inspect $\{x_{i_2}, \neg x_{i_2}\}$, if the number of their occurrences is greater than c, let c be the number of their occurrences. If $c > 3$, $\eta \notin \cup_{s=1}^{3}(3, s)\text{-}SAT_N$, this process completes. Otherwise, repeat this process for $\{x_{i_3}, \neg x_{i_3}\}, \cdots, \{x_{i_n}, \neg x_{i_n}\}$. If the result $c > 3$, $\eta \notin \cup_{s=1}^{3}(3, s)\text{-}SAT_N$; if $c \leq 3$, $\eta \in \cup_{s=1}^{3}(3, s)\text{-}SAT_N$. Clearly, this process takes polynomial time in the length of η.

We would like to remark that in the statement (5) above, if η has m clauses, then $\tilde{\eta}$ has at most $43m$ clauses as shown in [13]. This upper bound can be reduced from $43m$ to $31m$ by modifying Tovey's procedure as follows:

Step 1. Check if there is any variable with more than 4 occurrences in the expression. If there is no such variable, the process completes. Otherwise, go to Step 2.

Step 2. For convenience, we may assume that variable x appears in k clauses with $k > 4$. Other cases can be handled in a similar way. Create k new variables $x_1, \cdots, x_k$ and replace the ith occurrence of x with x_i, $i = 1, \cdots, k$. Create clauses $(x_i \vee \neg x_{i+1})$ for $i = 1, \cdots, k-1$ and clause $(x_k \vee \neg x_1)$. The clause $(x_i \vee \neg x_{i+1})$ implies that if x_i is false, x_{i+1} must be false as well. The cyclic structure of the clauses therefore forces the x_i to be either all true or all false. For each clause $(x_i \vee \neg x_{i+1})$, for $i = 1, \cdots, k-1$, and the clause $(x_k \vee \neg x_1)$, introduce new variable $y_i^{(x)}$, so that the clause becomes $(x_i \vee \neg x_{i+1} \vee \neg y_i^{(x)})$ or $(x_k \vee \neg x_1 \vee \neg y_k^{(x)})$. Now note that we can force each $y_i^{(x)}$ to be true by means of the clauses below in which $y_i^{(x)}$ appears only three times and other variables appear four times:

$$\begin{aligned}
&(y_i^{(x)} \vee a_{i,j}^{(x)} \vee b_{i,j}^{(x)}),\ j = 1, 2,\\
&(d_{i,j}^{(x)} \vee \neg a_{i,j}^{(x)} \vee \neg b_{i,j}^{(x)}), (d_{i,j}^{(x)} \vee \neg a_{i,j}^{(x)} \vee b_{i,j}^{(x)}), (d_{i,j}^{(x)} \vee a_{i,j}^{(x)} \vee \neg b_{i,j}^{(x)}),\ j = 1, 2,\\
&(\neg d_{i,1}^{(x)} \vee \neg d_{i,2}^{(x)} \vee y_i^{(x)}).
\end{aligned}$$

In order to make the above nine clauses satisfiable, $y_i^{(x)}$ must be true. The proof is the same as the proof in the first case of subprocess (3) of Theorem 1. Append the clause $(x_i \vee \neg x_{i+1} \vee \neg y_i^{(x)})$ for $i = 1, \cdots, k-1$, the clause $(x_k \vee \neg x_1 \vee \neg y_k^{(x)})$ and their associate nine clauses above to the modified expression. Note that the expression leaving this step is satisfiable if and only if the expression entered to this step is satisfiable. Go to Step 1.

Since Step 2 reduces one variable with occurrences greater than 4 each time and does not create any more such variable, and the original expression η has finite number of such variables, the process terminates. Since η is normal and the modified expression in each Step 2 stays normal, the result expression is normal as well. Clearly, if η has m clauses, the final expression has at most $m + 3m + 27m = 31m$ clauses, and the procedure transforming any $\eta \in (3, s)\text{-}SAT_N$ with $s > 4$ to $\tilde{\eta} \in (3, 4)\text{-}SAT_N$ takes polynomial time in the length of η. Now the proof of the Classification Theorem completes.

Corollary 1. $\cup_{s=1}^{4}(3, s)\text{-}SAT_N$ is $\mathcal{NP}$-complete.

Proof. First, we show that $\cup_{s=1}^{4}(3,s)\text{-}SAT_N$ is in $\mathcal{NP}$. A nondeterministic polynomial time Turing machine can guess a truth assignment to a given expression $\eta \in \cup_{s=1}^{4}(3,s)\text{-}SAT_N$ and accept if the assignment satisfies η. Now Corollary 1 follows directly from Theorem 1 and Theorem 2.

As in the proof of Theorem 1, we can show that $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$ is in $\mathcal{NP}$. From Section 2, for any $\eta \in \cup_{s=1}^{4}(3,s)\text{-}SAT$, there exists the polynomial time map ϕ_{map}, such that $\phi_{map}(\eta) = \tilde{\eta} \in \cup_{s=1}^{4}(3,s)\text{-}SAT(n)$ for some integer n which depends on η, and $\tilde{\eta}$ is satisfiable if and only if η is satisfiable. We can view ϕ_{map} as a polynomial time reduction from $\cup_{s=1}^{4}(3,s)\text{-}SAT$ to $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT(n)$. Furthermore, ϕ_{map} keeps the normalness from the definition. Thus, we can view ϕ_{map} as the polynomial time reduction from $\cup_{s=1}^{4}(3,s)\text{-}SAT_N$ to $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$ as well. Now we have the following:

Corollary 2. $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$ is $\mathcal{NP}$-complete.

We would like to give more detail to the Step 1 in the proof of Theorem 2, and rewrite it as an algorithm. That is, for any

$$\begin{aligned}\eta \;=\;& (y_{k_1} \vee y_{k_2} \vee y_{k_3}) \wedge (y_{k_4} \vee y_{k_5} \vee y_{k_6}) \wedge \cdots \wedge \\ & (y_{k_{3m+1}} \vee y_{k_{3m+2}} \vee y_{k_{3m+3}}) \in \cup_{s=1}^{4}(3,s)\text{-}SAT_N(n),\end{aligned}$$

the algorithm checks if each variable has less than 4 occurrences in η. If so, it returns *true*; otherwise, it returns *false*.

Algorithm 2.

```
for $i = 1, 2, \cdots, n$
    set $c = 0$;
    for $j = 1, 2, \cdots, 3(m+1)$
        if $e_i(y_{k_j}) = true$ or $false$, increase $c$ by 1;
        if $c > 3$, return $false$;
if $i = n$, return $true$.
```

The elementary steps of Algorithm 2 are setting $c = 0$, increasing c by 1, checking if $c > 3$ and evaluating $e_i(y_{k_j})$, for $i = 1, 2, \cdots, n$ and $j = 1, 2, \cdots, 3(m+1)$, and returning *true* or *false*. It is a polynomial time algorithm.

5. Aggressive Truth Assignments

In this section, we introduce the concept of aggressive truth assignment and endow the set of all aggressive truth assignments with a metric. We use the Classification Theorem to extend the definition of generalized truth assignment as follows: for any $\eta \in \cup_{s=1}^{4}(3,s)\text{-}SAT_N(n)$, (1) it evaluates η as a generalized truth assignment; (2) it checks if $\eta \in \cup_{s=1}^{3}(3,s)\text{-}SAT_N(n)$. We call this extended generalized truth assignment an aggressive truth assignment and, for convenience, we continue to use the same notation as the truth assignment. If an instance of $\cup_{s=1}^{4}(3,s)$-$SAT_N(n)$ is in $\cup_{s=1}^{3}(3,s)$-$SAT_N(n)$, the aggressive truth assignment $e_1e_2\cdots e_m$ just returns a $true$ value. For any $\eta \in \cup_{s=1}^{4}(3,s)$-$SAT_N(n)$, the aggressive truth assignment $e_1e_2\cdots e_m$ works in this way:

(1) It evaluates η as a generalized truth assignment as shown in Algorithm 1. If $e_1e_2\cdots e_m(\eta)$ $= true$, it returns a $true$ value and the process completes, otherwise it returns a $false$ value and goes to next subprocess.

(2) It checks if $\eta \in \cup_{s=1}^{3}(3,s)$-$SAT_N(n)$, using Algorithm 2. If so, it returns a $true$ value, otherwise it returns a $false$ value.

So if $e_1e_2\cdots e_m(\eta) = true$, then η is satisfiable. In other words, if $e_1e_2\cdots e_m(\eta) = false$, η must be an instance of the $\mathcal{NP}$-complete problem. The aggressive truth assignments catch all easily decidable problems under the sense of the Classification Theorem and decide each instance in those problems in polynomial time.

For any given two aggressive truth assignments $e_1^1e_2^1\cdots e_m^1$ and $e_1^2e_2^2\cdots e_m^2$, we define the composition $\varphi = (e_1^1e_2^1\cdots e_m^1)(e_1^2e_2^2\cdots e_m^2)$ as follows: for any $\eta \in \cup_{s=1}^{4}(3,s)$-$SAT_N(n)$,

(a) if $e_1^2e_2^2\cdots e_m^2(\eta) = true$, $\varphi(\eta) = true$ and skips the evaluation $e_1^1e_2^1\cdots e_m^1(\eta)$;

(b) if $e_1^2e_2^2\cdots e_m^2(\eta) = false$ and $e_1^1e_2^1\cdots e_m^1(\eta) = true$, $\varphi(\eta) = true$;

(c) if $e_1^2e_2^2\cdots e_m^2(\eta) = false$ and $e_1^1e_2^1\cdots e_m^1(\eta) = false$, $\varphi(\eta) = false$.

Thus, from the definition, if $\varphi(\eta) = true$, η is satisfiable. We can extend this definition to the composition of k aggressive truth assignments for $k \geq 2$.

Let

$$\begin{aligned}
\mathcal{TA}^1 &= \{a \mid a \text{ is an aggressive truth assignment}\} \\
\mathcal{TA}^k &= \{(a_1)(a_2)...(a_k) \mid a_1, a_2, ..., a_k \in \mathcal{TA}^1\}, \text{ where } k \geq 2, \text{ and} \\
\mathcal{TA}^\infty &= \cup_{k=1}^{\infty}\mathcal{TA}^k.
\end{aligned} \tag{5}$$

We introduce a metric in $\mathcal{TA}^\infty \times \mathcal{TA}^\infty$. The distance between two atomic truth assignments is defined as: $d(x_i^*, \neg x_j^*) = d(\neg x_j^*, x_i^*) = \frac{i+j}{2^{i+j+2}}$, and $d(x_i^*, x_j^*) = d(x_j^*, x_i^*) = d(\neg x_i^*, \neg x_j^*) = d(\neg x_j^*, \neg x_i^*) = \frac{|i-j|}{2^{i+j+2}}$ for all integers $i, j \geq 1$. Thus, $d(x_i^*, \neg x_i^*) = d(\neg x_i^*, x_i^*) = \frac{i}{2^{2i+1}}$, and $d(e_i, e_i) = 0$, for all integers $i \geq 1$. For any $e_1e_2\cdots e_m, e_1'e_2'\cdots e_m' \in \mathcal{TA}^1$, it is defined as

$$d(e_1e_2\cdots e_m, e_1'e_2'\cdots e_m') = \sum_{k=1}^{\infty} d(e_k, e_k'). \tag{6}$$

From the definition, for any $a_1, a_2, a_3 \in \mathcal{TA}^1$, we have

$$\begin{aligned}
&d(a_1, a_2) \geq 0, \\
&d(a_1, a_2) = 0 \text{ if and only if } a_1 = a_2, \\
&d(a_1, a_2) = d(a_2, a_1), \text{ and} \\
&d(a_1, a_3) \leq d(a_1, a_2) + d(a_2, a_3).
\end{aligned}$$

So $d : \mathcal{TA}^1 \times \mathcal{TA}^1 \to [0, \infty)$ is a metric and $(\mathcal{TA}^1, d)$ is a metric space. The above definitions can be extended to the following cases: $d(e_i, \cdot) = d(\cdot, e_i) = \frac{i}{2^{i+2}}$ for all integers $i \geq 1$ where $\cdot$ is the empty parameter. For any $e_1 e_2 \cdots e_m \in \mathcal{TA}^1$, $d(e_1 e_2 \cdots e_m, \cdot) = d(\cdot, e_1 e_2 \cdots e_m) = \sum_{k=1}^{\infty} d(e_k, \cdot) = \sum_{k=1}^{\infty} \frac{k}{2^{k+2}}$ where $\cdot$ is the empty parameter. For any $a_1, \cdots, a_m, b_1, \cdots, b_n \in \mathcal{TA}^1$,

$$d(a_1 \cdots a_m, b_1 \cdots b_n) = \begin{cases} \frac{1}{1^2} d(a_1, b_1) + \cdots + \frac{1}{m^2} d(a_m, b_m) + \\ \quad \frac{1}{(m+1)^2} d(b_{m+1}, \cdot) \ + \cdots + \frac{1}{n^2} d(b_n, \cdot) & \text{if } 0 < m < n, \\ \frac{1}{1^2} d(a_1, b_1) + \cdots + \frac{1}{n^2} d(a_n, b_n) + \\ \quad \frac{1}{(n+1)^2} d(a_{n+1}, \cdot) \ + \cdots + \frac{1}{m^2} d(a_m, \cdot) & \text{if } 0 < n < m, \text{ and} \\ \frac{1}{1^2} d(a_1, b_1) + \cdots + \frac{1}{n^2} d(a_n, b_n) & \text{if } 0 < m = n. \end{cases} \tag{7}$$

From the definition, we can verify the following:

Lemma 1. $d : \mathcal{TA}^\infty \times \mathcal{TA}^\infty \to [0, \infty)$ is a metric and $(\mathcal{TA}^\infty, d)$ is a metric space.

6. Pseudo-algorithms

Both $\cup_{s=1}^4 (3, s)\text{-}SAT_N$ and $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)\text{-}SAT_N(n)$ are $\mathcal{NP}$-complete from Corollary 1 and Corollary 2. We use these problems to study the $\mathcal{P}$ versus $\mathcal{NP}$ problem in the rest of this paper. Suppose that there are some polynomial time algorithms on $\cup_{s=1}^4 (3, s)\text{-}SAT_N$. They are also the polynomial time algorithms on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)\text{-}SAT_N(n)$. Let

$$\mathcal{A} = \{ f_\xi \mid f_\xi \text{ is a polynomial time algorithm on } \cup_{s=1}^4 \cup_{n=3}^\infty (3, s)\text{-}SAT_N(n) \}. \tag{8}$$

We prove that $\mathcal{A}$ is empty using the proof by contradiction in the following sections. Suppose that $\mathcal{A}$ is not empty.

For each aggressive truth assignment $a \in \mathcal{TA}^1$, we define

$$\mathcal{A}a = \{ f_\xi a \mid f_\xi \in \mathcal{A} \}, \tag{9}$$

and $f_\xi a$ as follows

$$f_\xi a(\eta) = \begin{cases} true & \text{if } a(\eta) = true, \\ f_\xi(\eta) & \text{if } a(\eta) \neq true, \end{cases} \tag{10}$$

for any $\eta \in \cup_{s=1}^4 (3, s)\text{-}SAT_N(n)$ and $n \geq 3$. If $a(\eta) = true$, η is satisfiable and the algorithm terminates. If $a(\eta) \neq true$, the algorithm applies f_ξ to η. Note that, for any $a, b \in \mathcal{TA}^1$, concerning not only final results but also computation processes or steps, we have $f_\xi a \neq f_\xi aa$, if $a \neq b$, then $f_\xi ab \neq f_\xi ba$.

Suppose that $\mathcal{A}a \subset \mathcal{A}$ for any $a \in \mathcal{TA}^1$. Choose an arbitrary $f \in \mathcal{A}$. Define the following sets:

$$\begin{aligned} < f >^0 &= \{f\} \\ < f >^k &= \{ f\alpha \mid \alpha \in \mathcal{TA}^k \} \text{ for } k = 1, 2, \cdots \text{ and,} \\ < f > &= \cup_{k=0}^\infty < f >^k . \end{aligned} \tag{11}$$

We have $< f >^k \bigcap < f >^l = \emptyset$ if $k \neq l$ and $< f >^k \neq \emptyset$ by the definition, and $< f >^k \subset \mathcal{A}$ for all integers $k \geq 0$ by the assumption.

For any $a_1, \cdots, a_m, b_1, \cdots, b_n \in \mathcal{TA}^1$, and $fa_1 \cdots a_m,\ fb_1 \cdots b_n \in < f >$, define

$$d(fa_1 \cdots a_m, fb_1 \cdots b_n) = \begin{cases} 0 & \text{if } m = n = 0, \\ d(a_1 \cdots a_m, b_1 \cdots b_n) & \text{if } m > 0 \text{ or } n > 0. \end{cases} \tag{12}$$

From Lemma 1, we can see that $d :< f > \times < f > \to [0, \infty)$ is a metric and $(< f >, d)$ is a metric space. For any $f\alpha_1 \in < f >^k$, $f\alpha_2 \in < f >^l$ and any $\beta \in \mathcal{TA}^m$ with $m \geq 1$, if $k \neq l$, $d(f\alpha_1\beta, f\alpha_2\beta) < d(f\alpha_1, f\alpha_2)$; if $k = l$, $d(f\alpha_1\beta, f\alpha_2\beta) = d(f\alpha_1, f\alpha_2)$. Thus, for any $\beta \in \mathcal{TA}^m$, the map $\beta :< f >^k \to < f >^{k+m}$ is equidistant. The metric d can be extended from $< f > \times < f >$ to $\mathcal{A} \times \mathcal{A}$ as follows: For any given $g_1, g_2 \in \mathcal{A}$, if $g_1 \notin < f >$ or $g_2 \notin < f >$ for any $f \in \mathcal{A}$, define $d(g_1, g_2) = 1$. For such g_1 and g_2, for any $\beta \in \mathcal{TA}^m$ with $m \geq 1$, $d(g_1\beta, g_2\beta) = d(g_1, g_2) = 1$.

A sequence $\{f_n\}$ of $< f >$ is called Cauchy if for any real number $\epsilon > 0$, there exists an integer $N > 0$, such that for all natural numbers $m, n > N$, $d(f_m, f_n) < \epsilon$.

Now we define a pseudo-algorithm ρ on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$ as follows: (1) ρ is not an algorithm on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, (2) for any $\eta \in \cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, ρ checks if $\eta \in \cup_{s=1}^3 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$ and decides any instance of $\cup_{s=1}^3 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, and (3) for any algorithm f_ξ on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, $f_\xi\rho$ is an algorithm on the same set. For any $m \geq 1$, any element of $\mathcal{AT}^m$ is a pseudo-algorithm on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$. In particular, any aggressive truth assignment is a pseudo-algorithm on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$.

For any algorithm φ on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, we define

$$\mathcal{A}\varphi = \{f_\xi\varphi \mid f_\xi \in \mathcal{A}\}, \tag{13}$$

and $f_\xi\varphi$ as follows

$$f_\xi\varphi(\eta) = \begin{cases} true & \text{if } \varphi(\eta) = true, \\ false & \text{if } \varphi(\eta) \neq true, \end{cases} \tag{14}$$

for any $\eta \in \cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, i.e., $f_\xi\varphi \equiv \varphi$ on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$.

7. Equivalence Classes

In practice, when implementing an algorithm, we usually break it to some processes. Each process has its subprocess, and each subprocess has its steps. Depending on the complexity of the algorithm, we may break it to more or less levels. For a given algorithm, if the inputs are the same, the outputs or the results are the same, and the implementations have the same sequence of steps. This sequence of steps is called an implementation sequence. We handle any pseudo-algorithm in a similar way.

Definition 1. A bijective map π from $\cup_{s=1}^{4} \cup_{n=3}^{\infty} (3,s)\text{-}SAT_N(n)$ to itself is ordered if $\pi(\cup_{s=1}^{4}(3,s)\text{-}SAT_N(n)) = \cup_{s=1}^{4}(3,s)\text{-}SAT_N(n)$ for all integers $n \geq 3$.

Definition 2. Algorithms φ_1, φ_2 are equivalent if there exists a bijective and ordered map π from $\cup_{s=1}^{4} \cup_{n=3}^{\infty} (3,s)\text{-}SAT_N(n)$ to itself, such that for any $\eta \in \cup_{s=1}^{4} \cup_{n=3}^{\infty} (3,s)\text{-}SAT_N(n)$, $\varphi_1(\eta)$ and $\varphi_2(\pi(\eta))$ have the same implementation sequences.

Definition 3. Pseudo-algorithms ρ_1, ρ_2 are equivalent if there exists a bijective and ordered map π from $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$ to itself, such that for any $\eta \in \cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$, $\rho_1(\eta)$ and $\rho_2(\pi(\eta))$ have the same implementation sequences.

If steps s_1 and s_2 are the same, we write $s_1 = s_2$. Suppose that σ_1 and σ_2 are algorithms or pseudo-algorithms on $\cup_{s=1}^{4} \cup_{n=3}^{\infty} (3,s)\text{-}SAT_N(n)$. If σ_1 and σ_2 are equivalent, we write $\sigma_1 \equiv \sigma_2$. The following remarks set up the rule to compare two implementation sequences.

Remark 1. In Algorithm 1, the following equations give all the same steps:
(1) $e_i(y_i) = e_i(y_i)$ and $\neg e_i(y_i) = \neg e_i(y_i)$ where $y_i = x_i$ or $\neg x_i$, for $i = 1, 2, \cdots, n$;
(2) $e_i(y_i) = \neg e_i(\neg y_i)$ where $y_i = x_i$ or $\neg x_i$, for $i = 1, 2, \cdots, n$;
(3) $e_i(x_j) = e_i(\neg x_j) = \neg e_i(x_j) = \neg e_i(\neg x_j)$ if $i \neq j$;
(4) {returning *true*} = {returning *true*} and {returning *false*} = {returning *false*}.

Remark 2. In Algorithm 2, the following equations give all the same steps:
(1) {setting $c = 0$} = {setting $c = 0$} and {increasing c by 1} = {increasing c by 1};
(2) {checking if $c > 3$} = {checking if $c > 3$};
(3) $e_i(y_i) = e_i(y_i)$ and $\neg e_i(y_i) = \neg e_i(y_i)$ where $y_i = x_i$ or $\neg x_i$, for $i = 1, 2, \cdots, n$;
(4) $e_i(y_i) = \neg e_i(\neg y_i)$ where $y_i = x_i$ or $\neg x_i$, for $i = 1, 2, \cdots, n$;
(5) $e_i(x_j) = e_i(\neg x_j) = \neg e_i(x_j) = \neg e_i(\neg x_j)$ if $i \neq j$;
(6) {returning *true*} = {returning *true*} and {returning *false*} = {returning *false*}.

Example 2. (1) The implementation sequence for $e_1e_2e_3e_4(\eta_1)$ in Example 1 is the following: $e_1(\neg x_1)$, $e_1(\neg x_2)$, $e_2(\neg x_2)$, $e_1(\neg x_1)$, $e_1(x_2)$, $e_2(x_2)$, $e_2(x_3)$, $e_3(x_3)$ and returning *false*.

(2) The implementation sequence for checking if $\eta_1 \in \cup_{s=1}^{3} \cup_{n=3}^{\infty} (3,s)\text{-}SAT_N(n)$ in Example 1 is the following: $c = 0$, $e_1(\neg x_1)$, increasing c by 1, checking if $c > 3$, $e_1(\neg x_2)$, $e_1(\neg x_3)$, $e_1(\neg x_1)$, increasing c by 1, checking if $c > 3$, $e_1(x_2)$, $e_1(x_3)$, $c = 0$, $e_2(\neg x_1)$, $e_2(\neg x_2)$, increasing c by 1, checking if $c > 3$, $e_2(\neg x_3)$, $e_2(\neg x_1)$, $e_2(x_2)$, increasing c by 1, checking if $c > 3$, $e_2(x_3)$, $c = 0$, $e_3(\neg x_1)$, $e_3(\neg x_2)$, $e_3(\neg x_3)$, increasing c by 1, checking if $c > 3$, $e_3(\neg x_1)$, $e_3(x_2)$, $e_3(x_3)$, increasing c by 1, checking if $c > 3$, $c = 0$, $e_4(\neg x_1)$, $e_4(\neg x_2)$, $e_4(\neg x_3)$, $e_4(\neg x_1)$, $e_4(x_2)$, $e_4(x_3)$ and returning *true*.

The equivalence of algorithms or pseudo-algorithms defined above is essentially a special case of Definition 3.2 in [7]. However, for the different purposes, the equivalence in [7] is much finer than ones in this paper.

Lemma 2. Any $a_1, a_2 \in \mathcal{TA}^1$ are equivalent.

Proof. Let $a_1 = e_1^1 e_2^1 \cdots e_n^1 \cdots$ and $a_2 = e_1^2 e_2^2 \cdots e_n^2 \cdots$. Define a map $\pi(a_1) = a_2$ as follows:

$\pi(e_i^1) = e_i^2$ for all intergers $i \geq 1$. Extend π to from $\cup_{s=1}^4 \cup_{n=3}^\infty (3,s)\text{-}SAT_N(n)$ to itself as follows:

$$\pi(y_i) = \begin{cases} y_i & \text{if } \pi(e_i^1) = e_i^1, \\ \neg y_i & \text{if } \pi(e_i^1) = \neg e_i^1, \end{cases} \tag{15}$$

where $y_i = x_i$ or $\neg x_i$, and

$$\pi(*) = \begin{cases} (& \text{if } * = (, \\ ) & \text{if } * =), \\ \vee & \text{if } * = \vee, \\ \wedge & \text{if } * = \wedge. \end{cases} \tag{16}$$

It is easy to verify that π is a bijective and ordered map from $\cup_{s=1}^4 \cup_{n=3}^\infty (3,s)\text{-}SAT_N(n)$ to itself and π^2 is the identical map.

From the construction of map π, Remark 1 and Remark 2, for any $\eta \in \cup_{s=1}^4 (3,s)\text{-}SAT_N(n)$, evaluating η and $\pi(\eta)$ respectively, a_1 and a_2 have the same implementation sequence, and checking if η and $\pi(\eta) \in \cup_{s=1}^3 (3,s)\text{-}SAT_N(n)$ respectively, a_1 and a_2 also have the same implementation sequence, i.e., $a_1(\eta)$ and $a_2(\pi(\eta))$ have the same implementation sequence. Thus, $a_1 \equiv a_2$.

The map π in Lemma 2 is uniquely determined by a_1 and a_2. On the other hand, from Definition 2, Remark 1 and Remark 2, any map π' which makes a_1 and a_2 equivalent is identical to π, i.e., we have the following:

Lemma 3. For any $a_1, a_2 \in \mathcal{TA}^1$, if map π' makes a_1 and a_2 equivalent, then π' is identical to π which is defined in Lemma 2.

Proof. We use proof by contradiction here. Let $a_1 = e_1^1 e_2^1 \cdots e_n^1 \cdots$ and $a_2 = e_1^2 e_2^2 \cdots e_n^2 \cdots$ be equivalent under π and π'. From the Definition 3, π' is a bijective and ordered map from $\cup_{s=1}^4 \cup_{n=3}^\infty (3,s)\text{-}SAT_N(n)$ to itself. We may assume that $\pi'(y_i) = y_i$ or $\pi'(y_i) = \neg y_i$, where $y_i = x_i$ or $\neg x_i$, for $i = 1, 2, \cdots$. Suppose that $\pi \neq \pi'$. There exists the minimum integer i_0 such that $\pi(x_{i_0}) \neq \pi'(x_{i_0})$. We can choose $\eta_0 \in \cup_{s=1}^4 (3,s)\text{-}SAT_N(n)$ for some integer n, such that the first clause of η_0 is $(x_{i_0} \vee y_l \vee y_k)$ where $y_l = x_l$ or $\neg x_l$ and $y_k = x_k$ or $\neg x_k$. Since a_1 and a_2 are equivalent under π and π', $e_{i_0}^1(x_{i_0}) = e_{i_0}^2(\pi(x_{i_0}))$ and $e_{i_0}^1(x_{i_0}) = e_{i_0}^2(\pi'(x_{i_0}))$. From Remark 1, $\pi(x_{i_0}) = \pi'(x_{i_0})$, this contradicts the assumption $\pi(x_{i_0}) \neq \pi'(x_{i_0})$. Thus, we have $\pi' = \pi$.

The map π making a_1 and b_1 equivalent is uniquely determined by a_1 and b_1. Let $a_1 = e_1 \cdots e_{k-1} e_k e_{k+1} \cdots$ and $a_1' = e_1 \cdots e_{k-1} \neg e_k e_{k+1} \cdots$, then a_1' and b_1 are equivalent under different map π'. It is interesting to know if the assumption $a_1 \equiv b_1$ and $a_2 \equiv b_2$ under the same map π can imply that $a_1 = a_2$ and $b_1 = b_2$. In general, it cannot imply that. For example, let $a_1 = x_1^* \neg x_2^* e_3 \cdots$, $a_2 = \neg x_1^* x_2^* e_3 \cdots$, $b_1 = \neg x_1^* x_2^* e_3 \cdots$ and $b_2 = x_1^* \neg x_2^* e_3 \cdots$. From the proof of Lemma 2, $a_1 \equiv b_1$ under map π, where $\pi(x_1) = \neg x_1$, $\pi(x_2) = \neg x_2$ and $\pi(x_i) = x_i$, for $i = 3, 4, \cdots$. However, $a_2 \equiv b_2$ under the same map π, $a_1 \neq a_2$ and $b_1 \neq b_2$.

Lemma 4. For any $a_1, a_2, \cdots, a_m, b_1, b_2, \cdots, b_m \in \mathcal{TA}^1$ with $m \geq 2$, $a_1a_2\cdots a_m \equiv b_1b_2\cdots b_m$ if and only if $a_i \equiv b_i$ under the same map π, for $i = 1, 2, \cdots, m$.

Proof. Suppose that $a_i \equiv b_i$ under the same map π. From the definition of composition of aggressive truth assignments and Definition 3, for any $\eta \in \cup_{s=1}^{4}(3,s)\text{-}SAT_N(n)$, $a_1a_2\cdots a_m(\eta)$ and $b_1b_2\cdots b_m(\pi(\eta))$ have the same implementation sequences. Thus, $a_1a_2\cdots a_m \equiv b_1b_2\cdots b_m$ under map π.

Suppose that $a_1a_2\cdots a_m \equiv b_1b_2\cdots b_m$ under map π. We want to prove that $a_i \equiv b_i$ under the same map π, for $i = 1, 2, \cdots, m$. By Definition 3 and the assumption, $a_m \equiv b_m$ under the same map π. For any $\eta \in \cup_{s=1}^{4}(3,s)\text{-}SAT_N(n)$ with $a_m(\eta) = b_m(\pi(\eta)) = true$, since a_m and b_m are generalized truth assignments, we can choose some clauses $\theta_1, \cdots, \theta_i$ such that $\eta' = \eta \wedge \theta_1 \wedge \cdots \wedge \theta_i \in \cup_{s=1}^{4}(3,s)\text{-}SAT_N(n')$ with $n' \geq n$ and $a_m(\eta') = b_m(\pi(\eta')) = false$. Without loss of generality, we may assume that $a_m(\eta) = b_m(\pi(\eta)) = false$. By the definition 3 and the assumption, $a_{m-1}(\eta)$ and $b_{m-1}(\pi(\eta))$ have the same implementation sequences, i.e., $a_{m-1} \equiv b_{m-1}$ under map π. We can apply this argument to k and get $a_k \equiv b_k$ under the same map π for $k = m-2, \cdots, 2, 1$. The proof is complete.

Corollary 3. For any $a_1, a_2, \cdots, a_m \in \mathcal{TA}^1$ with $m \geq 1$, $a_1a_2\cdots a_m \equiv a_1a_2\cdots a_m$ under the identical map.

Definition 2 and Definition 3 set up binary relations in the set of algorithms and pseudo-algorithms on $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$ respectively. It is easy to verify that these relations are equivalence relations. From Lemma 2, $\mathcal{TA}^1$ has one equivalence class. However, when $m \geq 2$, $\mathcal{TA}^m$ has infinitely many equivalence classes by Lemma 4. For any algorithm φ, φ_1 and φ_2 on $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$ and $a_1a_2\cdots a_m, b_1b_2\cdots b_m \in \mathcal{TA}^m$, if $\varphi_1 \equiv \varphi_2$ under map π_1, $a_1a_2\cdots a_m \equiv b_1b_2\cdots b_m$ under map π_2 and $\pi_1 = \pi_2$, then $\varphi_1a_1a_2\cdots a_m \equiv \varphi_2b_1b_2\cdots b_m$; however, if $\pi_1 \neq \pi_2$, $\varphi_1a_1a_2\cdots a_m \not\equiv \varphi_2b_1b_2\cdots b_m$. In particular, $\varphi a_1a_2\cdots a_m \not\equiv \varphi b_1b_2\cdots b_m$, unless $\varphi \equiv \varphi$ under map π_2. If $a_1a_2\cdots a_m \not\equiv b_1b_2\cdots b_m$, then $\varphi a_1a_2\cdots a_m \not\equiv \varphi b_1b_2\cdots b_m$.

Definition 4. A Cauchy sequence $\{f_n\}$ in $< f >^2$ is called regular if it satisfies the following conditions:
(1) $f_n = a_0a_n$ for $n = 1, 2, \cdots$;
(2) a_0 is an arbitrarily given generalized truth assignment;
(3) a_n and a_{n+1} are identical on the first n atomic truth assignments $e_1, e_2, \cdots, e_n$;
(4) a_n and a_{n+1} are the negative extensions of $e_1e_2\cdots e_{n+1}$ and $e_1e_2\cdots e_{n+2}$ respectively.

Note that the atomic truth assignment e_{n+1} of a_n usually takes $\neg x_{n+1}^*$, however, in some special cases, e_{n+1} can be x_{n+1}^*.

8. Proof of the Main Result

If f_ξ is a polynomial time algorithm on $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$ and a is an aggressive truth assignment, $f_\xi a$ is an algorithm on $\cup_{s=1}^{4}\cup_{n=3}^{\infty}(3,s)\text{-}SAT_N(n)$. From the previous sections,

$f_\xi a$ is a polynomial time algorithm as well. However, it is not clear if $f_\psi a \in \mathcal{A}$ or not for any $f_\psi \in \mathcal{A}$ and any aggressive truth assignment a. There are two cases:

(1) for any aggressive truth assignment a, $\mathcal{A}a \subseteq \mathcal{A}$;

(2) there is an aggressive truth assignment a_* such that $\mathcal{A}a_* \not\subset \mathcal{A}$.

We prove that neither case (1) nor case (2) is true in this section.

Case (1). From the assumption, any regular Cauchy sequence $\{f_n\}$ of $< f >^2$ is in $\mathcal{A}$ and coverges to a point f_ζ. We do not know if $f_\zeta \in \mathcal{A}$ or not, however, we can represent it as some algorithm on $\cup_{s=1}^4 \cup_{n=3}^\infty (3,s)\text{-}SAT_N(n)$. We may assume, by Definition 4, that $f_1 = fa_0a_1, f_2 = fa_0a_2, \cdots, f_n = fa_0a_n, \cdots$, where a_0 is the negative generalized truth assignment, a_n and a_{n+1} are identical on atomic truth assignments $e_1, e_2, \cdots, e_n$, and a_n and a_{n+1} are the negative extensions of $e_1e_2\cdots e_{n+1}$ and $e_1e_2\cdots e_{n+2}$ respectively. For any instance $\eta \in \cup_{s=1}^4 \cup_{n=3}^\infty (3,s)\text{-}SAT_N(n)$, there exists an integer n such that $\eta \in \cup_{s=1}^4 (3,s)\text{-}SAT_N(n)$. On the other hand, applying the polynomial time map ϕ_{map} to η, we get $\eta' = \phi_{map}(\eta) \in \cup_{s=1}^4 (3,s)\text{-}SAT_N(n)$. Since ϕ_{map} is the identical map in $\cup_{s=1}^4 (3,s)\text{-}SAT_N(n)$ for any integer $n \geq 3$, $\eta' = \eta$. For any given $\eta \in \cup_{s=1}^4 \cup_{n=3}^\infty (3,s)\text{-}SAT_N(n)$, we can find the integer n with $\eta \in \cup_{s=1}^4 (3,s)\text{-}SAT_N(n)$ in polynomial time. Thus, f_ζ can be represented as an algorithm on $\cup_{s=1}^4 \cup_{n=3}^\infty (3,s)\text{-}SAT_N(n)$ as follows: for any $\eta \in \cup_{s=1}^4 (3,s)\text{-}SAT_N(n)$ and any $n \geq 3$,

$$f_\zeta^*(\eta) = f_{n-2}(\eta) = fa_0a_{n-2}(\eta). \tag{17}$$

It is not difficult to see from (17) that f_ζ^* can be represented by finite information for each n. This algorithm is called characteristic representation of f_ζ. Since ϕ_{map} and f are polynomial time algorithms, a_0 and a_n are polynomial time pseudo-algorithms, f_ζ^* is a polynomial time algorithm.

Let $\mathcal{CS}$ be the set of all regular Cauchy sequences in $< f >^2$. We set up the equivalence relation in $\mathcal{CS}$ using Lemma 2, Lemma 3 and Lemma 4. In the following discussion, we just consider the equivalence classes of $\mathcal{CS}$. Since each aggressive truth assignment is defined by finite information, $\mathcal{TA}^1$ is countable, however, there is only one element in $\mathcal{TA}^1$ under the equivalence relation. In general, $< f >^1$ is countable under the equivalence relation. We want to prove that the equivalence classes of $\mathcal{CS}$ is uncountable. Suppose that the equivalence classes of $\mathcal{CS}$ is countable, and all equivalence classes of $\mathcal{CS}$ can be listed as

$$\begin{array}{rcl} \{f_n^1\} & = & fa_0a_1^1, fa_0a_2^1, \cdots, fa_0a_k^1, \cdots, \\ \{f_n^2\} & = & fa_0a_1^2, fa_0a_2^2, \cdots, fa_0a_k^2, \cdots, \\ & & \cdots, \\ \{f_n^k\} & = & fa_0a_1^k, fa_0a_2^k, \cdots, fa_0a_k^k, \cdots, \\ & & \cdots. \end{array} \tag{18}$$

Suppose that

$$a_1^1 \;=\; e_1^1e_2^1\cdots e_k^1\cdots,$$

$$\begin{aligned} a_2^2 &= e_1^2 e_2^2 \cdots e_k^2 \cdots, \\ &\cdots, \\ a_k^k &= e_1^k e_2^k \cdots e_k^k \cdots, \\ &\cdots. \end{aligned} \tag{19}$$

Define

$$\begin{aligned} a_1 &= \neg e_1^1 e_2 \neg x_3^* \cdots \neg x_k^* \neg x_{k+1}^* \cdots, \\ a_2 &= \neg e_1^1 \neg e_2^2 e_3 \neg x_4^* \cdots \neg x_k^* \neg x_{k+1}^* \cdots, \\ &\cdots, \\ a_k &= \neg e_1^1 \neg e_2^2 \cdots \neg e_k^k e_{k+1} \neg x_{k+2}^* \cdots, \\ &\cdots, \end{aligned} \tag{20}$$

where $e_m = x_m^*$ or $\neg x_m^*$ for $m \geq 2$, and

$$\{f_n\} = fa_0a_1, fa_0a_2, \cdots, fa_0a_n, \cdots. \tag{21}$$

We can adjust the aggressive truth assignments in (20) if necessary. For example, if $a_0 \equiv a_0$ and $a_1 \equiv a_1^1$ under the same map π_1, we can adjust $a_1 = \neg e_1^1 \neg e_2 \neg x_3^* \cdots \neg x_n^* \cdots$. From Lemma 2 and Lemma 3, $a_0 \equiv a_0$ and $a_1 \equiv a_1^1$, but under different maps. Similarly, if $a_0 \equiv a_0$ and $a_2 \equiv a_2^2$ under the same map π_2, we can adjust $a_2 = \neg e_1^1 \neg e_2^2 \neg e_3 \neg x_4^* \cdots \neg x_n^* \cdots$ such that $a_0 \equiv a_0$ and $a_2 \equiv a_2^2$ under different maps. This adjustification has no impact on the previous one, i.e., the claim $a_0 \equiv a_0$ and $a_1 \equiv a_1^1$ under different maps is still true. Clearly, $a_1, a_2, \cdots, a_k, \cdots$ are aggressive truth assignments by the construction. From the assumption, $\{f_n\}$ is in $\mathcal{A}$. It is not difficult to see that $\{f_n\}$ is a Cauchy sequence from definitions (6), (7) and (12), and satisfies all conditions in Definition 4. Thus, $\{f_n\} \in \mathcal{CS}$, however, from the construction, $fa_0a_1 \neq fa_0a_1^1, fa_0a_2 \neq fa_0a_2^2, \cdots, fa_0a_k \neq fa_0a_k^k, \cdots$, under the equivalence relation, i.e., $\{f_n\}$ is not in the list (18). This is a contradiction, which implies that $\mathcal{CS}$ is uncountable. Any element of $\mathcal{CS}$ has a characteristic representation which is a polynomial time algorithm on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, different elements have different characteristic representations. Therefore, there exist uncountably many algorithms in $\mathcal{A}$. This is absurd, since there are only countably many algorithms (see e.g. [8]). So case (1) is not true.

Case (2). From the assumption, there exist an $f_\lambda \in \mathcal{A}$ and an $a_* \in \mathcal{TA}^1$, such that $f_\lambda a_* \notin \mathcal{A}$. Since a_* is a polynomial time pseudo-algorithm and f_λ is a polynomial time algorithm on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, $f_\lambda a_*$ is a polynomial time algorithm on $\cup_{s=1}^4 \cup_{n=3}^\infty (3, s)$-$SAT_N(n)$, i.e., $f_\lambda a_* \in \mathcal{A}$. This is a contradiction. So Case 2 is not true, either.

Lemma 5. $\mathcal{A} = \emptyset$.

Proof. It follows directly from the above discussion.

We know that $\cup_{s=1}^4 (3, s)$-SAT_N is $\mathcal{NP}$-complete. Lemma 5 claims that there does not exist any polynomial time algorithm on $\cup_{s=1}^4 (3, s)$-SAT_N, so $\cup_{s=1}^4 (3, s)$-$SAT_N \notin \mathcal{P}$. We have the following result:

Theorem 3. $\mathcal{P} \neq \mathcal{NP}$.

We can apply the new argument to $2SAT$, however, we cannot get any contradiction, so we cannot change the status of $2SAT$. Since $2SAT \in \mathcal{P}$, if we classify $2SAT$, we obtain $2SAT$ itself or some classes in $2SAT$ which are also in $\mathcal{P}$. For any such classification, we can define an aggressive truth assignment $e_1e_2\cdots e_m$ as the following: for any $\eta \in 2SAT(n)$, (1) it evaluates η as a generalized truth assignment; (2) it checks that if η is satisfiable using Aspvall, Plass and Tarjan's algorithm [2]. Now for any instance $\eta \in 2SAT$, the aggressive truth assignment $e_1e_2\cdots e_m$ works in this way: (1) it evaluates η as a truth assignment, if $e_1e_2\cdots e_m(\eta) = true$, it returns a true value, otherwise it goes to next step; (2) it checks if η is satisfiable using Aspvall, Plass and Tarjan's algorithm [2]. If η is satisfiable, it returns a true value, otherwise it returns a false value. So $e_1e_2\cdots e_m(\eta) = true$ if and only if η is satisfiable. Since Aspvall, Plass and Tarjan's algorithm is linear time algorithm on $2SAT$, all aggressive truth assignments are linear time algorithms on $2SAT$ as well. Clearly, the pseudo-algorithm concept is suitable for $\cup_{s=1}^{4}(3,s)\text{-}SAT_N$, but not suitable for $2SAT$.

Under the assumption $\mathcal{A} \neq \emptyset$ and $a\mathcal{A} \subset \mathcal{A}$ for any $a \in \mathcal{TA}^1$, without using diagonalization method, we can prove that there exists a one to one map from a subset of $\mathcal{A}$ to all real numbers in $(0,1)$. This means that $\mathcal{A}$ is, at least, uncountable. However, the argument is tedious.

Acknowledgements

It is my pleasure to thank Dr. Gary R. Jensen for his encouragement. I am also very grateful to Dr. Yining Wang, who has contributed many ideas and comments. I greatly appreciate my wife, Ms. Yongmei Feng, and my children for their understanding, patience, support and love.

References

[1] S.Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computing Theory* 1(1):1-54, 2009.

[2] Aspvall, Bengt; Plass, Michael F.; Tarjan, Robert E. A Linear-time algorithm for testing the truth of certain quantified boolean formulas, *Information Processing Letters* 8(3):121-123, 1979.

[3] T.Baker, J.Gill, and R.Solovary. Relativizations of the P=?NP question. *SIAM J. Comput.*, 4:431-442, 1975.

[4] Cantor, Georg. Über eine Eigenschaft des Inbegriffes aller reelen algebraischen Zahlen, *Journal für die Reine und Angewandte Mathematik* 77, 258-262, 1874.

[5] Cantor, Georg. Über eine Elementare Frage der Mannigfaltigkeitslehre, *Jahresbericht der Deutschen Mathematiker-Vereinigung* 11, 75-78, 1890.

[6] Cook, Stephen A. The complexity of theorem-proving procedures, *Proc. 3rd Ann. ACM Symp. on Thoery of Computing, Association for Computing Machinery, New York*, 151-158, 1971.

[7] Gurevich, Yuri. Sequential abstract state machines capture sequential algorithms, *ACM Transactions on Computational Logic*, Vol.1 No.1, 77-111, 2000.

[8] Hein, James L. 2010, Discrete Structures, Logic, and Computability, 3rd edition. Jones and Bartlett Publishers: Sudbury, MA, section 14.1.1.

[9] Levin, Leonid A. Universal search problems (in Russian), *Problems of Information Transmission* 9(3): 265-266, 1973.

[10] Karp, Richard M. Reducibility among combinatorial problems, in R.E. Miller and J.W. Thatcher (eds.), *Complexity of Computer Computations, Plenum Press, New York*, 85-103, 1972.

[11] A. A. Razborov and S. Rudich. Natural proofs. *J. Comput. Sys. Sci.*, 55(1):24-35, 1997.

[12] Shamir, Adi. IP = PSPACE. Journal of the ACM, volume 39(4):869-877, 1992.

[13] Tovey, Craig A. A simplified satisfiability problem, *Discrete Appl. Math.* 8, 85-89, 1984.